Epic Technical Specification: Foundation & Infrastructure

Date: 2025-11-26
Author: BMAD Architect Agent
Epic ID: epic-1
Status: Ready for Implementation

Overview

Epic 1 establece la **base técnica hexagonal completa** para el Sistema de Enrutamiento y Gestión de Firmas Digitales. Este epic es foundational y prerequisito para todos los epics subsecuentes, implementando la infraestructura core que soportará 90 Functional Requirements y 47 Non-Functional Requirements del sistema bancario.

El epic entrega un proyecto Spring Boot 3 ejecutable con arquitectura hexagonal, PostgreSQL 15 con migrations versionadas, Kafka para event streaming, HashiCorp Vault para secrets management, y un environment de desarrollo local completamente dockerizado. Todos los componentes están configurados siguiendo compliance bancario (PCI-DSS, GDPR, SOC 2).

Objectives and Scope

In-Scope (8 Stories)

✅ **Story 1.1**: Proyecto Spring Boot 3 + Java 21 con estructura hexagonal (domain/, application/, infrastructure/)
✅ **Story 1.2**: PostgreSQL 15 + Flyway migrations con schema completo (6 tablas + UUIDv7 + JSONB + TDE)
✅ **Story 1.3**: Kafka cluster + Zookeeper + Schema Registry con topics y Avro schemas
✅ **Story 1.4**: HashiCorp Vault integration para secret management (no hardcoded credentials)
✅ **Story 1.5**: Domain models puros (SignatureRequest aggregate, ValueObjects, Enums) - zero dependencies externas
✅ **Story 1.6**: JPA entities + repository adapters (infrastructure layer) + entity mappers
✅ **Story 1.7**: REST API foundation con OpenAPI 3.1, OAuth2 JWT security, global exception handler
✅ **Story 1.8**: Docker Compose con todos los servicios para desarrollo local (1-command startup)

Out-of-Scope

❌ **Business logic implementation** (viene en Epic 2: Signature Orchestration)
❌ **Provider integrations** (viene en Epic 3: Multi-Provider Integration)
❌ **Resilience patterns** (viene en Epic 4: Circuit Breaker & Fallback)
❌ **Event publishing logic** (viene en Epic 5: Event-Driven Architecture)
❌ **Admin Portal UI** (viene en Epic 6-7: Admin Portal)
❌ **Production deployment configs** (Kubernetes manifests se crean después)

Success Criteria for Epic 1

Este epic se considera exitoso cuando:

1. ✅ Desarrollador puede clonar repo, ejecutar `docker-compose up -d` y `mvn spring-boot:run`, y ver app iniciando sin errores
2. ✅ `/actuator/health` retorna HTTP 200 con todos los componentes UP (PostgreSQL, Kafka, Vault)
3. ✅ Flyway migrations crean schema completo con 6 tablas
4. ✅ Domain models (agregados, VOs) compilan sin imports de Spring/JPA (verificado con ArchUnit)
5. ✅ Swagger UI accessible en `/swagger-ui.html` con endpoints documentados
6. ✅ Integration tests con Testcontainers pasan (PostgreSQL, Kafka)

System Architecture Alignment

Este epic implementa las **capas foundation** definidas en `docs/architecture/02-hexagonal-structure.md`:

Hexagonal Architecture Layers

```
com.bank.signature/
├── domain/                    # Story 1.5: Pure business logic
│   ├── model/
│   │   ├── aggregate/SignatureRequest
│   │   ├── entity/SignatureChallenge
│   │   └── valueobject/TransactionContext, Money
│   ├── service/               # Interfaces (implementations in Epic 2)
│   └── port/
│       ├── inbound/           # Use case interfaces
│       └── outbound/          # Repository interfaces
│
├── application/               # Epic 2+: Use case implementations
│
```

```
└── infrastructure/            # Stories 1.6, 1.7: Adapters
    ├── adapter/
    │   ├── inbound/rest/      # Story 1.7: REST controllers
    │   └── outbound/
    │       ├── persistence/   # Story 1.6: JPA repositories
    │       ├── provider/      # Epic 3: Provider adapters
    │       └── event/         # Epic 5: Kafka publishers
    └── config/                # Stories 1.2-1.4: Spring configs
        ├── PersistenceConfig
        ├── KafkaConfig
        └── VaultConfig
```

Technology Stack (from `docs/architecture/01-system-overview.md`)

Component	Technology	Version	Story
Backend Framework	Spring Boot	3.2.0	1.1
Language	Java	21	1.1
Build Tool	Maven	3.9+	1.1
Database	PostgreSQL	15	1.2
Migration Tool	Flyway	9.x	1.2
Event Streaming	Kafka	3.6 (Confluent)	1.3
Schema Registry	Confluent Schema Registry	7.5	1.3
Secret Management	HashiCorp Vault	1.15	1.4
Connection Pool	HikariCP	(Spring Boot default)	1.2
API Documentation	SpringDoc OpenAPI	2.x	1.7
Security	Spring Security OAuth2	6.x	1.7
Testing	Testcontainers	1.19	1.2, 1.3, 1.6
Container Runtime	Docker Compose	v2	1.8

Database Schema (from `docs/architecture/03-database-schema.md`)

Story 1.2 implementa estas 6 tablas core:

1. **`signature_request`** - Aggregate root
   - PK: `id` (UUIDv7)
   - Columns: `customer_id` (pseudonymized), `transaction_context` (JSONB), `status`, `active_challenge_id`, `created_at`, `updated_at`, `expires_at`
   - Indexes: GIN en JSONB, B-tree en FK
2. **`signature_challenge`** - Entity
   - PK: `id` (UUIDv7)
   - FK: `signature_request_id`
   - Columns: `channel_type`, `provider`, `provider_challenge_id`, `provider_proof`, `status`, `sent_at`, `responded_at`, `expires_at`, `error_code`
3. **`routing_rule`** - Configuration
   - PK: `id` (UUIDv7)
   - Columns: `name` (UNIQUE), `priority`, `condition` (SpEL expression), `target_channel`, `enabled`, `created_by`, `updated_by`
   - Index: Priority ASC WHERE enabled=true
4. **`connector_config`** - Provider configuration
   - PK: `id` (UUIDv7)
   - Columns: `provider` (UNIQUE), `enabled`, `config` (JSONB), `vault_path`, `degraded_mode`, `error_rate`, `last_health_check`
5. **`outbox_event`** - Outbox pattern (Epic 5)
   - PK: `id` (UUIDv7)
   - Columns: `aggregate_id`, `aggregate_type`, `event_type`, `payload` (JSONB), `payload_hash`, `created_at`, `published_at`
6. **`audit_log`** - Immutable audit trail (Epic 8)
   - PK: `id` (UUIDv7)
   - Partitioned by month (PostgreSQL table partitioning)
   - Columns: `entity_type`, `entity_id`, `action`, `actor`, `changes` (JSONB), `ip_address`, `created_at`

Kafka Topics (from `docs/architecture/04-event-catalog.md`)

Story 1.3 crea estos topics:

- `signature.events`: Main event stream
  - Partitions: 12
  - Replication: 3
  - Retention: 7 days
  - Compression: snappy
- `signature.events.dlq`: Dead letter queue
  - Partitions: 3
  - Retention: 30 days

Detailed Design

Services and Modules

Story 1.1 estructura el proyecto Maven en módulos siguiendo hexagonal architecture:

Module/Package	Responsibility	Dependencies	Owner Story
`domain/model/aggregate`	Aggregate roots (SignatureRequest)	Zero external deps	1.5
`domain/model/entity`	Entities (SignatureChallenge, RoutingRule)	Zero external deps	1.5
`domain/model/valueobject`	Value objects (TransactionContext, Money)	Zero external deps	1.5
`domain/model/enums`	Domain enums (SignatureStatus, ChannelType)	Zero external deps	1.5
`domain/service`	Domain service interfaces	Zero external deps	1.5
`domain/port/inbound`	Use case interfaces	Zero external deps	1.5
`domain/port/outbound`	Repository & provider interfaces	Zero external deps	1.5
`domain/exception`	Domain exceptions	Zero external deps	1.5
`application/usecase`	Use case implementations (Epic 2)	domain/	Epic 2

Module/Package	Responsibility	Dependencies	Owner Story
`infrastructure/adapter/outbound/persistence`	JPA repos + mappers	domain/, JPA	1.6
`infrastructure/adapter/inbound/rest`	REST controllers	domain/, Spring Web	1.7
`infrastructure/config`	Spring configurations	Spring Boot	1.2–1.7

Inputs/Outputs:

- **Input (Story 1.1)**: Developer runs `mvn archetype:generate` → generates base project structure
- **Output (Story 1.8)**: `mvn clean install` succeeds, `docker-compose up -d` runs all services, app starts on port 8080

Data Models and Contracts

SignatureRequest Aggregate (Story 1.5)

```
package com.bank.signature.domain.model.aggregate;

@Value
@Builder
public class SignatureRequest {
    UUID id;                                    // UUIDv7
    String customerId;                          // Pseudonymized
    TransactionContext transactionContext;      // Immutable VO
    SignatureStatus status;                     // Enum: PENDING, CHALLENGE_SENT,
SIGNED, FAILED, EXPIRED
    UUID activeChallengeId;                     // Nullable
    List<SignatureChallenge> challenges;        // 1-N relationship
    List<RoutingEvent> routingTimeline;         // Audit trail
    Instant createdAt;
    Instant updatedAt;
    Instant expiresAt;

    // Business methods (to be implemented in Epic 2)
    public SignatureChallenge createChallenge(ChannelType channel) { ... }
    public void completeSignature(SignatureChallenge challenge) { ... }
    public void abort(AbortReason reason) { ... }
    public void expire() { ... }
}
```

TransactionContext Value Object (Story 1.5)

```
package com.bank.signature.domain.model.valueobject;

@Value
public class TransactionContext {
    String transactionId;
    String transactionType;      // WIRE_TRANSFER, P2P_TRANSFER, etc.
    Money amount;
    String riskLevel;            // HIGH, MEDIUM, LOW
    Map<String, Object> customFields;  // JSONB flexibility
    Instant timestamp;

    // Immutable - no setters
    public String toJson() { ... }
    public static TransactionContext fromJson(String json) { ... }
}
```

SignatureChallenge Entity (Story 1.5)

```
package com.bank.signature.domain.model.entity;

@Value
@Builder
public class SignatureChallenge {
    UUID id;
    UUID signatureRequestId;
    ChannelType channelType;     // SMS, PUSH, VOICE, BIOMETRIC
    String provider;              // TWILIO, PUSH_SERVICE, etc.
    String providerChallengeId;
    String providerProof;         // Non-repudiation receipt
    ChallengeStatus status;       // PENDING, SENT, COMPLETED, FAILED, EXPIRED
    Instant sentAt;
    Instant respondedAt;
    Instant expiresAt;
    String errorCode;
    String rawResponse;
}
```

JPA Entity Mapping (Story 1.6)

```
package com.bank.signature.infrastructure.adapter.outbound.persistence.entity;

@Entity
@Table(name = "signature_request")
public class SignatureRequestEntity {
    @Id
```

```
    private UUID id;

    @Column(name = "customer_id", nullable = false)
    private String customerId;

    @Type(JsonBinaryType.class)
    @Column(name = "transaction_context", columnDefinition = "jsonb", nullable =
false)
    private String transactionContext;  // JSON string

    @Enumerated(EnumType.STRING)
    @Column(name = "status", nullable = false)
    private SignatureStatus status;

    @OneToMany(mappedBy = "signatureRequest", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<SignatureChallengeEntity> challenges = new ArrayList<>();

    @Column(name = "created_at", nullable = false)
    private Instant createdAt;

    @Column(name = "updated_at", nullable = false)
    private Instant updatedAt;

    @Column(name = "expires_at", nullable = false)
    private Instant expiresAt;

    // Getters/setters
}
```

APIs and Interfaces

Port Interfaces (Story 1.5)

Outbound Port - Repository:

```
package com.bank.signature.domain.port.outbound;

public interface SignatureRequestRepository {
    void save(SignatureRequest request);
    Optional<SignatureRequest> findById(UUID id);
    List<SignatureRequest> findByCustomerId(String customerId);
    List<SignatureRequest> findExpired(Instant now);
}
```

Inbound Port - Use Case (interface only en Story 1.5, impl en Epic 2):

```java
package com.bank.signature.domain.port.inbound;

public interface StartSignatureUseCase {
    SignatureResponse start(CreateSignatureRequest request);
}
```

REST API Endpoints (Story 1.7)

Base Configuration:

- Base Path: `/api/v1`
- Authentication: OAuth2 JWT (Bearer token)
- Content-Type: `application/json`
- OpenAPI Spec: `/swagger-ui.html`

Health Endpoint (Story 1.7 - functional immediately):

```
GET /actuator/health
Response 200 OK:
{
  "status": "UP",
  "components": {
    "db": { "status": "UP", "details": { "database": "PostgreSQL",
"validationQuery": "isValid()" } },
    "kafka": { "status": "UP" },
    "vault": { "status": "UP" }
  }
}
```

Signature Endpoints (scaffolded in Story 1.7, implemented in Epic 2):

```
POST /api/v1/signatures
Headers:
  Authorization: Bearer <jwt>
  Idempotency-Key: <uuid>
Request Body: CreateSignatureRequest
Response: 201 Created

GET /api/v1/signatures/{id}
Response: 200 OK SignatureResponse
```

Admin Endpoints (scaffolded in Story 1.7):

```
POST /api/v1/admin/rules
PUT /api/v1/admin/rules/{id}
GET /api/v1/admin/rules
DELETE /api/v1/admin/rules/{id}
```

Error Response Contract:

```
{
  "code": "VALIDATION_ERROR",
  "message": "Invalid transaction context",
  "details": [
    { "field": "transactionContext.amount.value", "error": "Must be greater than
0" }
  ],
  "timestamp": "2025-11-26T10:30:00Z",
  "traceId": "550e8400-e29b-41d4-a716-446655440000"
}
```

Workflows and Sequencing

Story Execution Sequence

Epic 1 stories **deben** ejecutarse en este orden (dependencies):

```
Story 1.1 (Project Bootstrap)
    ↓
Story 1.2 (PostgreSQL + Flyway) ← depends on 1.1
    ↓
Story 1.3 (Kafka + Schema Registry) ← depends on 1.1
    ↓
Story 1.4 (Vault Integration) ← depends on 1.1
    ↓
Story 1.5 (Domain Models) ← depends on 1.1
    ↓
Story 1.6 (JPA Entities) ← depends on 1.2, 1.5
    ↓
Story 1.7 (REST API Foundation) ← depends on 1.1, 1.5
    ↓
Story 1.8 (Docker Compose) ← depends on 1.2, 1.3, 1.4
```

Application Startup Sequence (Story 1.7 onwards)

```
1. Spring Boot Application starts
   ↓
2. Vault Connection (Story 1.4)
   - Connects to Vault at localhost:8200
   - Loads secrets: db-password, twilio-api-key
```

```
   ↓
3. Flyway Migrations (Story 1.2)
   - Executes V1__initial_schema.sql
   - Creates 6 tables with constraints
   ↓
4. HikariCP Connection Pool (Story 1.2)
   - Initializes 5 min, 20 max connections
   - Timeout: 2s
   ↓
5. Kafka Producer (Story 1.3)
   - Connects to localhost:9092
   - Registers Avro schemas in Schema Registry
   ↓
6. Spring Security (Story 1.7)
   - Configures OAuth2 Resource Server
   - JWT validation with RSA public key
   ↓
7. REST Controllers (Story 1.7)
   - Registers endpoints
   - OpenAPI spec generated
   ↓
8. Health Check Available
   - GET /actuator/health returns UP
```

Non-Functional Requirements

Performance

NFR-P1: Application startup time < 30 seconds (Story 1.7)

- **Measurement**: Time from `mvn spring-boot:run` to "Started Application in X seconds"
- **Target**: < 30s on laptop (8GB RAM, SSD)

NFR-P2: Flyway migration execution < 5 seconds (Story 1.2)

- **Measurement**: Time to execute V1___initial_schema.sql
- **Target**: < 5s (empty database, 6 tables + indexes)

NFR-P3: Database connection pool ready < 2 seconds (Story 1.2)

- **Target**: HikariCP initialization < 2s

NFR-P4: Health endpoint response < 100ms (Story 1.7)

- **Measurement**: GET /actuator/health latency
- **Target**: P99 < 100ms

Security

NFR-S1: Zero hardcoded credentials (Story 1.4)

- **Validation**: Grep codebase for passwords → zero matches
- **Implementation**: All credentials in Vault (`secret/signature-router/*`)

NFR-S2: TLS encryption at-rest (PostgreSQL TDE) (Story 1.2)

- **Requirement**: PostgreSQL `ssl = on` in configuration
- **Validation**: `SHOW ssl;` returns `on`

NFR-S3: OAuth2 JWT authentication (Story 1.7)

- **Requirement**: All `/api/v1/**` endpoints require valid JWT
- **Validation**: Request without Authorization header → HTTP 401

NFR-S4: Domain layer has zero Spring dependencies (Story 1.5)

- **Validation**: ArchUnit test fails if `domain/` imports `org.springframework.*`

Reliability/Availability

NFR-A1: Database connection pool resilience (Story 1.2)

- **Requirement**: HikariCP leak detection threshold 60s
- **Configuration**: `spring.datasource.hikari.leak-detection-threshold=60000`

NFR-A2: Kafka producer timeout handling (Story 1.3)

- **Requirement**: Producer send timeout 1.5s
- **Configuration**: `spring.kafka.producer.properties.request.timeout.ms=1500`

NFR-A3: Graceful shutdown (Story 1.7)

- **Requirement**: Application waits 30s for in-flight requests before shutdown
- **Configuration**: `spring.lifecycle.timeout-per-shutdown-phase=30s`

Observability

NFR-O1: Structured JSON logging (Story 1.7)

- **Implementation**: Logback + Logstash encoder
- **Format**: `{"timestamp":"...", "level":"INFO", "logger":"...", "message":"...", "traceId":"..."}`

NFR-O2: Health endpoint with component details (Story 1.7)

- **Requirement**: `/actuator/health` shows db, kafka, vault status
- **Response**: HTTP 200 if all UP, HTTP 503 if any DOWN

NFR-O3: Actuator endpoints exposed (Story 1.7)

- **Exposed**: `/actuator/health`, `/actuator/info`, `/actuator/metrics`, `/actuator/prometheus`
- **Not exposed**: `/actuator/shutdown`, `/actuator/env` (security risk)

Dependencies and Integrations

Maven Dependencies (Story 1.1)

Core Spring Boot:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>3.2.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

Database & Migrations (Story 1.2):

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
    <version>9.22.0</version>
</dependency>
```

Kafka (Story 1.3):

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
    <groupId>io.confluent</groupId>
    <artifactId>kafka-avro-serializer</artifactId>
    <version>7.5.0</version>
</dependency>
```

Vault (Story 1.4):

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-vault-config</artifactId>
</dependency>
```

OpenAPI (Story 1.7):

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.2.0</version>
</dependency>
```

Utilities:

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.github.f4b6a3</groupId>
    <artifactId>uuid-creator</artifactId>
    <version>5.3.2</version>
</dependency>
```

Testing (Story 1.6):

```
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>postgresql</artifactId>
    <version>1.19.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>kafka</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.tngtech.archunit</groupId>
    <artifactId>archunit-junit5</artifactId>
    <version>1.1.0</version>
    <scope>test</scope>
</dependency>
```

External Service Integrations (Story 1.8)

Docker Compose Services:

Service	Image	Port	Story
PostgreSQL	`postgres:15-alpine`	5432	1.2
Kafka	`confluentinc/cp-kafka:7.5.0`	9092	1.3
Zookeeper	`confluentinc/cp-zookeeper:7.5.0`	2181	1.3
Schema Registry	`confluentinc/cp-schema-registry:7.5.0`	8081	1.3

Service	Image	Port	Story
Vault	`hashicorp/vault:1.15`	8200	1.4

Acceptance Criteria (Authoritative)

Epic-Level Acceptance Criteria

AC-E1-1: Developer can clone repository and run `mvn clean install` successfully

- **Validation**: Exit code 0, no compilation errors

AC-E1-2: Developer can start all infrastructure with `docker-compose up -d`

- **Validation**: All 5 services (PostgreSQL, Kafka, Zookeeper, Schema Registry, Vault) show status "Up"

AC-E1-3: Application starts with `mvn spring-boot:run` and reaches "Started Application" log

- **Validation**: Application log shows "Started SignatureRouterApplication in X seconds"

AC-E1-4: Health endpoint returns HTTP 200 with all components UP

- **Validation**: `curl http://localhost:8080/actuator/health` returns `{"status":"UP"}`

AC-E1-5: Database schema contains 6 tables with correct structure

- **Validation**: `SELECT count(*) FROM information_schema.tables WHERE table_schema='public'` returns 6

AC-E1-6: Domain layer has zero Spring dependencies

- **Validation**: ArchUnit test `domainShouldNotDependOnSpring()` passes

AC-E1-7: Swagger UI accessible and shows documented endpoints

- **Validation**: Browser navigate to `http://localhost:8080/swagger-ui.html` shows OpenAPI UI

AC-E1-8: Integration tests with Testcontainers pass

- **Validation**: `mvn verify` runs integration tests, exit code 0

Story-Level Acceptance Criteria

Ver `docs/epics.md` Epic 1 para AC detallados por story (8 stories x ~4-6 AC cada una)

Traceability Mapping

AC ID	Epic Requirement	Story	Component/API	Test Strategy
AC-E1-1	Compilable project	1.1	pom.xml, package structure	`mvn clean install`
AC-E1-2	Infrastructure services	1.2, 1.3, 1.4, 1.8	docker-compose.yml	`docker-compose ps`
AC-E1-3	Application startup	1.1, 1.7	SignatureRouterApplication.java	Log assertion
AC-E1-4	Health checks	1.7	HealthController, actuator	Integration test
AC-E1-5	Database schema	1.2	V1___initial_schema.sql	SQL query test
AC-E1-6	Hexagonal purity	1.5	domain/ package	ArchUnit test
AC-E1-7	API documentation	1.7	OpenApiConfig, @Operation annotations	Manual browser test
AC-E1-8	Integration testing	1.2, 1.3, 1.6	*IT.java classes	`mvn verify`

Risks, Assumptions, Open Questions

Risks

RISK-1: Docker Compose memory requirements may exceed laptop limits

- **Impact**: High (blocks local development)
- **Probability**: Medium (Kafka + PostgreSQL + Vault ~4GB RAM)
- **Mitigation**: Document minimum system requirements (8GB RAM, prefer 16GB). Provide lightweight alternative (H2 in-memory for quick tests).

RISK-2: Vault dev mode loses secrets on restart

- **Impact**: Medium (developers need to re-configure)
- **Probability**: High (dev mode is stateless)
- **Mitigation**: Document in README. Provide init script to seed common secrets. Production uses Kubernetes auth with persistence.

RISK-3: Flyway migration failures leave database in inconsistent state

- **Impact**: High (requires manual intervention)
- **Probability**: Low (migrations tested in CI)
- **Mitigation**: Use Flyway validation. Implement rollback scripts for V1 (DROP TABLE IF EXISTS). Test migrations against empty + pre-populated databases.

RISK-4: UUIDv7 function compatibility across PostgreSQL versions

- **Impact**: Medium (migration fails on PostgreSQL <13)
- **Probability**: Low (using PostgreSQL 15)
- **Mitigation**: Document minimum PostgreSQL version. Include version check in migration.

Assumptions

ASSUMPTION-1: Developers have Docker Desktop installed and running

- **Validation**: Document in prerequisites

ASSUMPTION-2: Java 21 JDK available on developer machines

- **Validation**: `mvn -version` check, fail fast with clear error if Java <21

ASSUMPTION-3: Port 8080, 5432, 9092, 8081, 8200 available (not in use)

- **Validation**: Document port requirements, provide script to check port availability

ASSUMPTION-4: Internet connectivity available for Maven dependency download

- **Validation**: Use Maven Wrapper to ensure consistent Maven version

Open Questions

QUESTION-1: Should we use Spring Boot Devtools for auto-restart during development?

- **Decision Needed**: By Story 1.1
- **Impact**: Developer UX improvement

QUESTION-2: Do we need separate Maven profiles for dev/test/prod?

- **Decision Needed**: By Story 1.1
- **Current**: Single profile, externalized config via environment variables

QUESTION-3: Should Kafka topics be auto-created or pre-created?

- **Decision Needed**: By Story 1.3
- **Recommendation**: Pre-create in migration for explicit control (production requires this)

QUESTION-4: How do we handle Vault secret rotation in development?

- **Decision Needed**: By Story 1.4
- **Current Thinking**: Manual rotation in dev, automated in prod (out of scope for Epic 1)

Test Strategy Summary

Test Levels

Test Level	Scope	Framework	Example	Story
Unit Tests	Domain models, pure logic	JUnit 5 + Mockito	`SignatureRequestTest`	1.5
Integration Tests	Repository adapters, DB	JUnit 5 + Testcontainers	`SignatureRepositoryIT`	1.6
API Tests	REST controllers	MockMvc + Testcontainers	`SignatureControllerIT`	1.7
Architecture Tests	Hexagonal purity	ArchUnit	`HexagonalArchitectureTest`	1.5
Contract Tests	OpenAPI spec	Spring Cloud Contract	(Future - Epic 2+)	N/A

Test Coverage Targets

- **Domain Layer**: 90% line coverage (pure business logic)
- **Application Layer**: 80% line coverage (use cases)
- **Infrastructure Layer**: 70% line coverage (adapters)
- **Overall Project**: 75% minimum

Key Test Scenarios

Story 1.2 (Database):

- ✅ Flyway migration creates 6 tables
- ✅ UUIDv7 function generates sortable UUIDs
- ✅ JSONB columns store/retrieve complex objects
- ✅ Constraints prevent invalid data (CHECK, FK, UNIQUE)

Story 1.5 (Domain Models):

- ✅ SignatureRequest aggregate enforces invariants (1 active challenge)
- ✅ TransactionContext is immutable (no setters)
- ✅ Value objects implement equals/hashCode correctly
- ✅ Domain exceptions inherit from DomainException base

Story 1.6 (JPA Repositories):

- ✅ Save SignatureRequest → persisted with all challenges
- ✅ FindById round-trip (save → find → assert equality)
- ✅ JSONB mapping (Java object → JSON → Java object)
- ✅ Cascade delete (delete request → challenges deleted)

Story 1.7 (REST API):

- ✅ Endpoint without JWT → HTTP 401
- ✅ Endpoint with valid JWT → HTTP 200
- ✅ Invalid JSON → HTTP 400 with field errors
- ✅ OpenAPI spec generation (Swagger UI accessible)

Story 1.8 (Docker Compose):

- ✅ `docker-compose up -d` starts all services
- ✅ Application connects to all services (health checks pass)
- ✅ `docker-compose down -v` cleanup works

Continuous Integration

GitHub Actions Workflow (to be created):

```
name: Epic 1 - Foundation CI
```

```yaml
on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-java@v3
        with:
          java-version: '21'
      - name: Build with Maven
        run: mvn clean verify
      - name: Run ArchUnit tests
        run: mvn test -Dtest=HexagonalArchitectureTest
      - name: Upload coverage to Codecov
        uses: codecov/codecov-action@v3
```

Implementation Notes

Development Workflow

1. **Start Infrastructure**: `docker-compose up -d`
2. **Run Application**: `mvn spring-boot:run`
3. **Run Tests**: `mvn test` (unit), `mvn verify` (integration)
4. **View API Docs**: http://localhost:8080/swagger-ui.html
5. **Check Health**: http://localhost:8080/actuator/health

Story Development Order

Week 1:

- Story 1.1: Project Bootstrap (1 day)
- Story 1.2: PostgreSQL + Flyway (1 day)
- Story 1.5: Domain Models (2 days)

Week 2:

- Story 1.3: Kafka Infrastructure (1 day)
- Story 1.4: Vault Integration (1 day)
- Story 1.6: JPA Entities (2 days)

Week 3:

- Story 1.7: REST API Foundation (2 days)
- Story 1.8: Docker Compose (1 day)
- Integration testing & cleanup (2 days)

Total Effort: ~3 weeks (1 developer)

Success Metrics

After Epic 1 completion:

- ✅ 0 compilation errors
- ✅ 0 failing tests
- ✅ > 75% code coverage
- ✅ 0 critical SonarQube issues
- ✅ Application startup < 30s
- ✅ All health checks GREEN
- ✅ README with clear setup instructions

Epic Status: READY FOR STORY DRAFTING
Next Step: Run `create-story` workflow for Story 1.1 (Project Bootstrap)

This technical specification is authoritative for Epic 1 implementation. All stories must align with decisions documented here. Deviations require SM approval and tech spec update.